CardSpace-OpenID Integration for CardSpace Users

Haitham S. Al-Sinani and Chris J. Mitchell

Technical Report
RHUL–MA–2011–12
24 May 2011

Department of Mathematics
Royal Holloway, University of London
Egham, Surrey TW20 0EX, England

http://www.rhul.ac.uk/mathematics/techreports

Abstract

Whilst the growing number of identity management systems have the potential to reduce the threat of identity attacks, major deployment problems remain because of the lack of interoperability between such systems. In this paper we propose a novel, simple scheme to provide interoperability between two of the most widely discussed identity management systems, namely CardSpace and OpenID. In this scheme, CardSpace users are able to obtain an assertion token from an OpenID-enabled identity provider, the contents of which can be processed by a CardSpace-enabled relying party. The scheme, based on a browser extension, is transparent to OpenID providers and to the CardSpace identity selector, and only requires minor changes to the operation of a CardSpace-enabled relying party. We specify its operation and also describe an implementation of a proof-of-concept prototype. Additionally, security and operational analyses are provided.

Keywords: CardSpace, OpenID, Interoperation, Browser Extension

1 Introduction

In an attempt to simplify management of identities and mitigate identity-oriented attacks, a number of identity management systems (e.g. CardSpace, OpenID, Liberty, etc.) have been proposed [3]. An identity provider (IdP) in such a system supplies a user agent (UA) with an authentication token that can be consumed by a particular relying party (RP). Whilst one RP might solely support CardSpace, another might only support OpenID. Therefore, to make these systems available to the largest possible group of users, effective interoperability between such systems is needed. In this paper we investigate a case involving a CardSpace-enabled RP, an OpenID-enabled IdP (also referred to as an OpenID provider (OP)), and a UA that is CardSpace-enabled. The goal is to develop a client-based approach to integration that is as transparent as possible to IdPs, RPs and identity selectors.

We consider CardSpace-OpenID interoperation because of OpenID's wide adoption by technology-leading organisations (see section 2.2.1). Complementing this, the wide use of Windows, recent versions of which incorporate CardSpace, means that enabling interoperation between the two systems is likely to be of significance for large numbers of identity management users and service providers. CardSpace-OpenID interoperation is also attractive since both schemes support the exchange of user attributes.

The remainder of the paper is organised as follows. Section 2 gives an overview of CardSpace and OpenID, and section 3 presents the integration

scheme. In section 4 we provide an operational analysis and, in section 5, we describe a prototype implementation. Section 6 reviews related work and, finally, section 7 concludes the paper.

2 CardSpace and OpenID

2.1 CardSpace

2.1.1 Introduction

CardSpace provides a secure and consistent way for users to control and manage personal data, to review personal data before sending it to a website, and to verify the identity of visited websites. It also enables websites to obtain data from users, e.g. to support user authentication and authorisation.

Digital identities are represented to users as Information Cards (or InfoCards). There are two types of InfoCards: personal (self-issued) cards and managed cards, issued by remote IdPs. Personal cards are created by users themselves, and the claims listed in such an InfoCard are asserted by the self-issued identity provider (SIIP) that co-exists with the CardSpace identity selector on the user machine. InfoCards do not contain sensitive information, but instead carry metadata indicating the types of personal data associated with this identity, and from where assertions regarding this data can be obtained. The data referred to by personal cards is stored on the user machine, whereas the data referred to by a managed card is held by the IdP that issued it [2, 4, 11].

By default, CardSpace is supported by Internet Explorer (IE) from version 7 onwards. Extensions to other browsers, such as Firefox[1] and Safari[2], also exist. An updated version, CardSpace 2.0 Beta 2, was released, although Microsoft announced in early 2011 that it will not ship; instead Microsoft has released a technology preview of U-Prove[3]. In this paper we refer throughout to the CardSpace version that is shipped by default as part of Windows Vista and Windows 7, that is available as a free download for XP and Server 2003, and which has been approved as an OASIS standard [10].

[1] `https://addons.mozilla.org/en-US/firefox/addon/openinfocard-identity-selector/`

[2] `http://www.hccp.org/safari-plug-in.html`

[3] `http://blogs.msdn.com/b/card/archive/2011/02/15/beyond-windows-cardspace.aspx`

2.1.2 Personal Cards

The scheme proposed here uses CardSpace personal cards to make information provided by OPs available to CardSpace RPs via the selector.

The selector allows a user to create a personal card and populate its fields with self-asserted claims. CardSpace restricts the contents of personal cards to non-sensitive data in the form of 14 editable claim types, namely *First Name, Last Name, Email Address, Street, City, State, Postal Code, Country/Region, Home Phone, Other Phone, Mobile Phone, Date of Birth, Gender* and *Web Page*. Data inserted in personal cards is stored in encrypted form on the user machine. At the time of creation, a card ID and a card master key are created and stored by the selector.

Using Personal Cards When using personal cards, CardSpace adopts the following protocol. We describe the protocol for the case where the RP does not employ a security token service (STS), a service responsible for token management [9].

1. UA $\rightarrow$ RP. HTTP/S request: GET (login page).

2. RP $\rightarrow$ UA. HTTP/S response. A login page is returned containing the CardSpace-enabling tags in which the RP security policy is embedded.

3. User $\rightarrow$ UA. The RP page offers the option to use CardSpace; selecting this option activates the selector, which is passed the RP policy. If this is the first time that this RP has been contacted, the selector will display the identity of the RP and give the user the option to either proceed or abort the protocol.

4. Selector $\rightarrow$ InfoCards. The selector, after evaluating the RP policy, highlights InfoCards matching the policy and greys out the rest. InfoCards previously used for this RP are displayed in the upper half of the selector screen.

5. User $\rightarrow$ selector. The user chooses a personal card. (Alternatively, the user could create and choose a new personal card). The user can preview the card (with its associated claims) to ensure that they are willing to release the claim values. Of the claims specified in an InfoCard, only those requested in the RP policy will be passed to the requesting RP.

6. Selector $\rightleftharpoons$ SIIP. The selector creates and sends a SAML-based Request Security Token (RST) to the SIIP, which responds with a SAML-based Request Security Token Response (RSTR).

7. UA $\rightarrow$ RP. The RSTR is passed to the UA, which forwards it to the RP.

8. RP $\rightarrow$ user. The RP validates the token, and, if satisfied, grants access.

Private Personal Identifiers (PPIDs) The PPID is an identifier linking a specific InfoCard to a particular RP [4]. When a user first uses a personal card at a particular RP, CardSpace generates a card-site-specific PPID by combining the card ID with data taken from the RP certificate, and a card-site-specific signature key pair by combining the card master key with data taken from the RP certificate. The RP domain/IP address is used if no RP certificate is available.

Since the PPID and key pair are RP-specific, the PPID does not function as a global user identifier, helping to enhance user privacy and reduce the impact of PPID compromise. The selector displays a shortened version of the PPID to protect against social engineering attacks and improve readability.

When a user first registers with an RP, the RP retrieves the PPID and the public key from the received SAML security token, and stores them. If a personal InfoCard is re-used at a site, the supplied security token will contain the same PPID and public key as used previously, and will be signed using the corresponding private key. The RP compares the received PPID and public key with its stored values, and verifies the digital signature.

The PPID could be used on its own as a shared secret to authenticate a user to an RP. However, it is recommended that the associated (public) signature verification key, as held by the RP, should also always be used to verify the signed security token to provide a more robust authentication method [4].

2.2 OpenID

2.2.1 Introduction

OpenID is an open and decentralised user authentication scheme supporting remote single sign-on to multiple websites using a single digital identity. As of December 2009, OpenID has been widely adopted, with more than one billion OpenIDs on the Internet and approximately nine million sites

enabling OpenID consumer support. OPs include Google, Facebook, and Microsoft.

OpenID 2.0 (and some OpenID 1.1 implementations) use(s) two types of user identifier: URLs and XRIs (Extensible Resource Identifiers). A user could adopt a self-owned URL, e.g. a home page, or register a (new) URL at an OP.

2.2.2 Operational Protocol

Two 'major' OpenID versions have been released: OpenID 1.1 [14], and OpenID 2.0 [13]; fortunately v2.0 is backward compatible with v1.1. We next describe the OpenID protocol, covering the main differences between the two 'major' versions.

Before the protocol run, a user will typically have previously registered an OpenID identifier with an OP.

1. UA $\rightarrow$ RP. HTTP/S request: GET (login page).
2. RP $\rightarrow$ UA. HTTP/S response. A login page is returned containing an OpenID login form.
3. User $\rightarrow$ UA. The user enters their OpenID identifier into the OpenID form, and submits it.
4. RP: OP discovery. The RP uses the user-supplied OpenID identifier to discover the user's OP, as follows.
   - **HTML-based discovery (OpenID 1.1/2.0)**. The RP requests an HTML document identified by the user's OpenID URL; such a document contains the information necessary to discover the required OP.
   - **XRDS-based discovery (OpenID 2.0)**. The RP requests an XRDS document containing the information necessary to discover the required OP. If the user's OpenID identifier is an:
     - XRI, the RP will retrieve an XRDS document identified by the user-supplied XRI; and
     - URL, the RP will use the Yadis protocol [12] to retrieve an XRDS document; if this fails, the RP will revert to HTML-based discovery.
5. RP $\rightleftharpoons$ OP (optional). The RP and OP agree a shared secret key to be used for a specified period of time by the OP and RP to MAC-protect

and verify subsequent protocol messages. Note that this request-response process, known as the 'association' mode, is transparent to the user, and requires the two parties to be able to store the secret.

6. RP-OP interaction. The RP and OP can communicate in either 'checkid_immediate' mode, involving direct RP-OP communications without user interaction, or 'checkid_setup' mode, where the user is interactively involved in RP-OP communications. The 'checkid_setup' mode is more commonly used; indeed if 'checkid_immediate' mode fails, the scheme typically reverts to 'checkid_setup' mode. If 'checkid_immediate' mode is being used, the RP directly sends the OP an OpenID authentication request, and the OP directly replies with an OpenID authentication response; step 9 then takes place. However, in 'checkid_setup' mode, the RP redirects the user to the OP with an OpenID authentication request[4], and step 7 follows.

7. OP $\rightleftharpoons$ user. If necessary, the OP authenticates[5] the user. If successful, the OP constructs an OpenID assertion token, including user credentials/attributes, a freshly-generated nonce[6], a current time-stamp, and a MAC computed on the token. If a shared key was agreed in step 5, the OP uses it to generate the MAC; otherwise the OP employs an internally-generated MAC key. The OP requests permission to send the assertion token to the requesting RP.

8. OP $\rightarrow$ UA $\rightarrow$ RP. The OP redirects the user back to the RP with a positive or negative OpenID authentication response, depending on whether or not the user granted permission in step 7.

9. RP $\rightarrow$ user. The RP validates the MAC-protected OpenID authentication response, and, if satisfied, grants access. The validation process includes verifying that the nonce has not been seen before, the time-stamp is sufficiently current, and the MAC is valid. The RP adds the received nonce to a list for use in future verifications. The RP uses the time-stamp to discard responses that are are 'too old', thus limiting the period of time for which received nonces must be kept. If a shared secret was previously agreed (see step 5), the RP uses its copy to verify

[4]OpenID requests and responses are typically sent embedded in URLs (alternatively they could be sent in HTML forms).

[5]Note that the authentication method used is not constrained by OpenID.

[6]Although mandatory in OpenID 2.0 (to prevent replay attacks), use of nonces is not mandatory in OpenID 1.1.

the MAC. If a secret was not agreed, the RP must make an extra request to the OP to verify the MAC, typically via a TLS/SSL channel. This request-response process is known as the 'check_authentication' mode, and is adopted in the integration scheme.

Note that the use of SSL/TLS on the OP-client and RP-client channels is strongly recommended. For additional security, the RP can add a freshly-generated nonce to its authentication request, which the OP must include in the authentication response.

2.2.3 Attribute Exchange

OpenID supports a range of methods for attribute exchange, including the Simple Registration OpenID Extension (SREG) [8], which allows the exchange of attributes of nine specified types (namely *nickname, email, fullname, dob, gender, postcode, country, language*), and Attribute Exchange (AX) [6], which supports the transfer of arbitrary data. Both SREG and AX are supported by the scheme described below.

3 The Integration Scheme

We now describe the novel scheme. The parties involved are a CardSpace-enabled RP, a CardSpace-enabled UA (e.g. a suitable web browser), an OP, and a browser extension implementing the protocol described below.

3.1 Requirements

The scheme has the following operational requirements.

- The user must have an existing relationship with both a CardSpace RP and an OP (thus the OP will have a means of authenticating the user). The RP must trust the OP for the purposes of user authentication.
- Prior to, or during, use of the integration protocol, the user must create a CardSpace personal card, referred to here as an IDcard. This IDcard must contain the following data items in specific fields (the choice of which is implementation-specific): the user's OpenID identifier; a predefined sequence of characters (e.g. 'OpenID') used to trigger the browser extension (see section 4.3) and indicate which OpenID version to use; and the URL of the OP.

- The CardSpace-enabled RP must not employ an STS. Instead, the RP must express its security policy using HTML/XHTML, and interactions between the selector and the RP must be based on HTTP/S via a web browser (a simpler and probably more common scenario for selector-RP interactions). This is because the scheme uses a browser extension, and is thus incapable of managing the necessary communications with an STS.

- The CardSpace-enabled RP must be prepared to accept an unsigned 'CardSpace-like' SAML token which includes both OP-asserted attributes and the digitally-signed SIIP-issued RSTR containing the card-RP-specific PPID.

3.2 Protocol Operation

The protocol operates as follows (a summary of the protocol is shown in figure 1). Steps 1, 2, and 4–7 are the same as steps 1, 2, and 3–6, respectively, of the personal card protocol given in section 2.1.2.

3. Browser extension → UA. The extension performs the following steps.

   (a) It scans the login page to detect whether the RP website supports CardSpace; if so, it proceeds, otherwise it terminates.

   (b) It examines the RP policy to check whether the use of personal cards is acceptable. If so, it proceeds; otherwise it terminates, giving CardSpace the opportunity to operate normally.

   (c) It keeps a local copy of any RP-requested claims.

   (d) It modifies the RP policy to include the types of claim employed in the IDcard. For example, if the user's OpenID identifier is stored in the *web page* field of the IDcard, then it must ensure that the RP security policy includes the *web page* claim. Note that adding the claim types to the RP policy ensures that the token supplied by the SIIP contains the values of these claims, which can then be processed by the browser extension; otherwise these values would not be available to the browser extension.

   (e) It determines the communication protocol (HTTP or HTTPS) in use.

8. Selector→ browser extension. Unlike in the 'standard' case, the RSTR does not reach the RP; instead the extension intercepts it and tem-

porarily stores it. If the RP uses HTTP[7], the extension uses the contents of the RSTR to construct an OpenID authentication request[8], which it forwards to the appropriate OP, having discovered its address from the RSTR.

If the RP uses HTTPS, the browser extension:

(a) asks the user to enter his/her OpenID identifier and thus uses the user-supplied OpenID to perform OP discovery (see section 2.2.2); and

(b) constructs an OpenID authentication request (precisely as in the HTTP case), which it forwards to the discovered OP.

Note that in both cases (i.e. HTTP and HTTPS) the format of the OP authentication request will depend on the version of OpenID being used (see below). Note also that in both cases the more commonly used OpenID 'checkid_setup' mode is adopted; the 'checkid_immediate' mode is not supported as it requires direct (background) RP-IdP communication without any user interaction.

9. OP $\rightleftharpoons$ user. If necessary, the OP authenticates the user. If successful, the OP requests permission to send the OpenID assertion token (see step 7 of the OpenID protocol given in section 2.2.2) to the designated RP return-page[9].

10. OP $\rightarrow$ UA $\rightarrow$ RP. The OP redirects the UA back to the RP return-page with a positive or negative OpenID authentication response[10], depending on whether or not the user granted permission in step 9.

11. Browser extension $\rightarrow$ UA. The browser extension verifies the MAC-protected OpenID authentication response by interacting with the OP

[7]Note that the protocol operates slightly differently depending on whether the RP uses HTTP or HTTPS. This is because, if HTTPS is used, then the selector will encrypt the RSTR message using the site's public key, and the browser extension does not have access to the corresponding private key. Hence, it will not know whether to trigger the integration protocol, and will be unable to obtain the user's OpenID identifier; such issues do not occur if HTTP is used since the selector will not encrypt the RSTR.

[8]This request will indicate the RP-requested user attributes which are to be asserted by the OP. The browser extension will know what they are since they were stored by it in step 3c.

[9]Note that the designated return-page, chosen by the browser extension in step 8, is the address to which the OP authentication response must be returned in step 10.

[10]The RP will receive the OP-issued token unchanged (embedded in the URL); however it is assumed that the RP will ignore it because of its inability to process the token.

using the 'check_authentication' mode via a TLS/SSL channel (see section 2.2.2). If the verification succeeds, it constructs a CardSpace-compatible SAML token (see below), and forwards it to the RP. If the verification fails, the browser extension informs the user and terminates.

12. RP → user. The RP verifies the SAML token (including verifying the RSTR signature, PPID, nonce, time-stamps, etc.), and, if satisfied, grants access.

The detailed operation of steps 8 and 11 is dependent on the OpenID version in use. For example, the authentication request name-space field (openid.ns) must be set to '`http://specs.openid.net/auth/2.0`' for OpenID 2.0, and one of absent, '`http://openid.net/signon/1.1`' or '`http://openid.net/signon/1.0`' for OpenID 1.1. Similarly, the field '`openid.ns.sreg=http://openid.net/extensions/sreg/1.1`' [8] is added to the authentication request when requesting identity attributes using the SREG extension in OpenID 2.0.

Observe that the (unsigned) SAML security token created by the browser extension in step 11 will include the user attributes as asserted by the OP and the digitally-signed SIIP-issued RSTR (which contains the PPID), allowing the RP to verify the SIIP signature (see also sections 4.6 and 4.7).

4 Discussion and Analysis

4.1 Defeating Phishing

The scheme mitigates the risk of phishing. This is because the redirect to the OP[11] is initiated by the browser extension and not by the RP, i.e. the RP cannot redirect the user to an OP of its choosing. By contrast, in OpenID a malicious RP could redirect a user to a fake OP, which might capture the user credentials.

4.2 Client-side Integration

IdPs/RPs may not accept the burden of supporting two identity management systems simultaneously, unless there is a significant financial incentive. Currently, major Internet players do not provide any means of interoperation between identity management systems. As a result, a client-side technique

[11] In HTTP mode the OP address is retrieved from the IDcard as entered by the user.

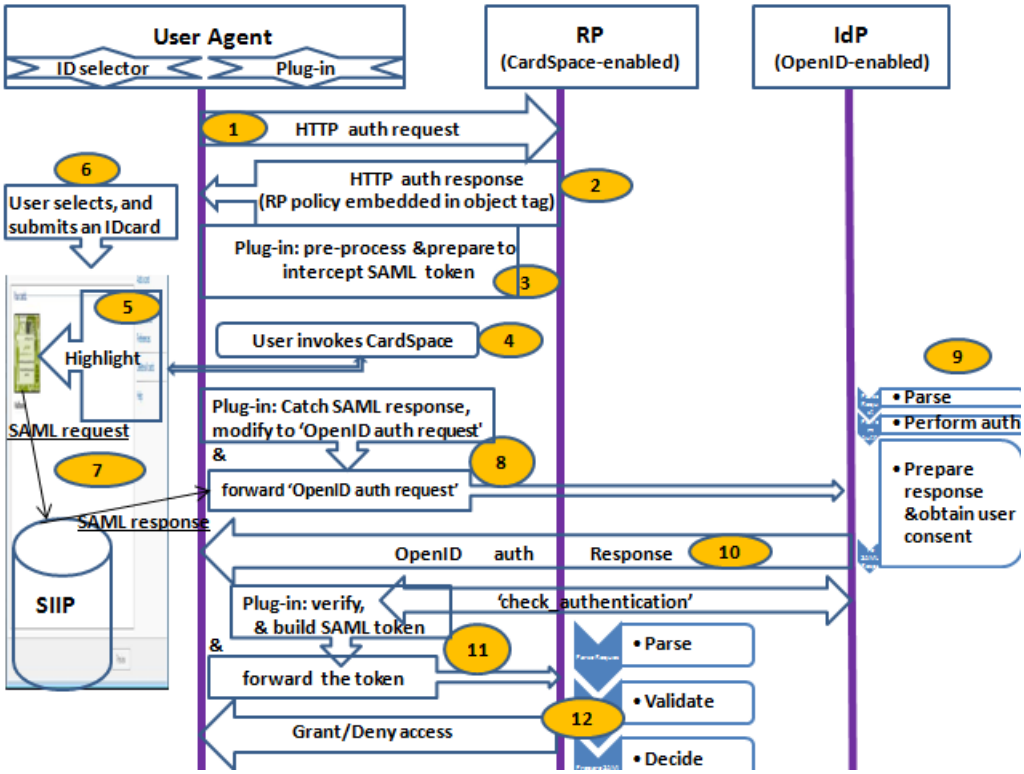

Figure 1: Protocol steps

for supporting interoperation could be practically useful. Supporting interoperation at the client also means that the performance of the server is not affected, since the integration overhead is handled by the client.

4.3 Triggering the Browser Extension

The scheme specified in section 3.1 (like the prototype implementation) uses a trigger sequence in a specific field of an IDcard. This trigger sequence is also employed to indicate to the browser extension which OpenID version to run. However, other approaches could be used, e.g. the browser extension could start whenever CardSpace is triggered. In such a case, when a user submits an IDcard, the browser extension could offer the user two options (based on HTML forms or JavaScript pop-up boxes): to continue to use CardSpace as usual, or to activate the integration scheme.

This approach gives a greater degree of user control, and hence imple-

ments Microsoft's first identity law [4, 11]. In addition, giving user control over whether the browser extension runs or not would enable 'normal' use of CardSpace. However, it is potentially a little inconvenient, since it would require users to always choose whether or not to use the integration software. Nevertheless, this effect could be mitigated if the user's choice could be stored.

4.4 IDcard Contents

A typical OpenID authentication request to an OP includes the user-supplied OpenID identifier, an RP return-page to which the OP must issue the authentication response, and a list of requested attributes. The RP must, of course, also know the OP address. In the protocol described in section 3.2 the user's OpenID identifier and the OP URL are specified in the IDcard[12]. The following alternative approaches avoid the need to store this data in the IDcard.

- The browser extension could prompt the user to enter the OpenID identifier that they wish to use, after they have submitted an IDcard, e.g. as part of step 8 in section 3.2. This approach would be inconvenient, since the user would have to enter the identifier every time, unless it could be remembered.
- The browser extension could keep an internal list of the widely used OP service URLs, enabling it to deduce which OP it needs to contact from the user's OpenID identifier. This would potentially maximise user transparency, but could give rise to storage issues and operational problems, e.g. in the case where an OP is not in the list. The latter issue could be addressed by prompting the user to enter the URL of an unknown OP, which the browser extension could then add to its internal list for future use.
- The browser extension could discover the OP from the user-supplied OpenID identifier, e.g. by fetching an HTML document from the URL of this identifier. This approach is vulnerable to phishing attacks and requires extra round trips.

4.5 Direct Communication

The prototype browser extension described in section 5 is currently not capable of making a direct (background) HTTP/S request to the OP (i.e. not via

[12]The RP return-page is transparently computed by the browser extension itself.

a browser). This could be performed using AJAX; indeed the OpenID specifications state that '...the authentication scheme plays nicely with AJAX-style setups' [13, 14]. However, AJAX is restricted by the 'same origin policy' for security reasons [15].

To address this problem, the browser extension performs an HTTP/S redirect whenever direct communication is needed, e.g. when the OpenID 'check_authentication' mode is activated. It then reads the data returned by the OP[13], redirects the user back to the previous page, and continues with the protocol. As a result, users will experience a relatively speedy (duplex) redirect, but will not be actively involved. Indeed, this is precisely what happens in the prototype implementation of the verification process for the OP-issued token in step 11 of section 3.2, and so such a redirect seems unlikely to be a major usability problem.

4.6 OP User Authentication

The SAML token created by the browser extension in step 11 of section 3.2 could be extended to contain an additional field to indicate that the user has been authenticated by a specified OP (as well as when and how). Of course, the RP would need to be modified to be able to process such an extra field, although this is likely to be relatively straightforward.

This authentication statement could also include the original token generated by the OP. Since this is a MAC-protected token, verifying it would give the RP added guarantees about the user authenticity. If this is required, and since the OP will only support a single validation of a security token[14], the browser extension must skip the verification process and send the OpenID token unchanged[15] to the CardSpace-enabled RP.

4.7 Security Considerations

The unsigned SAML token generated by the browser extension in step 11 of section 3.2 (referred to here as the 'user token') includes the PPID, the user attributes as asserted by the OP, the signed SIIP-issued RSTR, and (optionally) the MAC-protected OP-issued token. The RP compares the SIIP-asserted PPID (and the public key) in the user token with its stored values and verifies the digital signature (see section 2.1.2). The RP can

[13] URL query parameters or hidden form variables could be used to maintain state.

[14] To prevent replay attacks, OPs only issue a single valid verification for each request with the same nonce value [13].

[15] The OpenID token should be sent in an authentication statement contained within the SAML token, to allow RPs to choose whether or not to process it.

thus authenticate the user, link the user to his/her account, and consume the OP-asserted attributes, e.g. for authorisation purposes. In addition, an RP can optionally also verify the MAC in the OP-issued token, which is embedded unchanged in the user token. However, for the RP to be able to validate the MAC, the browser extension must skip the verification process (see section 4.6) and the RP must initiate on-line interaction with the OP via the 'check_authentication' mode.

A malicious entity cannot fabricate a user token to masquerade as a legitimate party since it will not have access to three key token components: the PPID; the SIIP-signed RSTR, which is only issued if the appropriate InfoCard is selected on the correct platform; and the MAC-protected OP-issued token, which is only issued if the genuine user has been authenticated by the OP. In addition, nonces and time-stamps are used to prevent replay attacks, and RPs can also employ IP address validation. As mentioned in section 2.2.1, the use of SSL/TLS is strongly recommended when using OpenID.

Note that, in protocol step 4, the selector identifies the RP to the user and indicates whether or not they have visited that particular RP before; if the user is visiting this RP for the first time, CardSpace requests the user's permission to proceed[16] (see section 2.1.2). This helps to support mutual authentication since the user and the RP are both identified to each other.

In addition to user authentication, the scheme also strengthens OpenID against phishing (see section 4.1). Finally note that the scheme allows the user attributes to be stored at the OP; this has potential security advantages over storing the attributes on the user machine, as is currently the case with CardSpace SIIP-issued attributes.

4.8 Attribute Mapping

As stated in sections 2.1.2 and 2.2.3, CardSpace personal cards currently support fourteen editable attributes, whereas the OpenID SREG extension only supports nine attribute types. The prototype described in section 5 uses the mapping in Table 1 to convert between the attribute types.

The OpenID SREG extension also supports *language* and *timezone* attributes, which have no corresponding attribute types in CardSpace personal cards.

The protocol specified in section 3.2 could also be used to support transfer of arbitrary data between OPs and CardSpace RPs using the AX exten-

[16] This offers a security advantage by comparison with 'native' OpenID, which does not identify the RP to the user.

Table 1: CardSpace-OpenID attribute mapping

CardSpace personal cards	OpenID SREG extension
givenname	nickname
surname	fullname
emailaddress	email
dateofbirth	dob
gender	gender
postalcode	postcode
country	country

sion (see section 2.2.3); however, this has not yet been prototyped.

5 Prototype Realisation

We next give details of a prototype implementation of the scheme. The description applies to both OpenID 1.1 and OpenID 2.0. The prototype uses the OpenID 'checkid_setup' mode, operating with the SREG extension.

The prototype is coded in JavaScript, chosen because its wide adoption should simplify the task of porting the prototype to a range of other browsers. It uses the Document Object Model (DOM) to inspect and manipulate HTML pages and XML documents. The JavaScript code is executed using a C#-driven browser helper object (BHO), a Dynamic-link library (DLL) module designed as a plug-in for IE. Once installed, the BHO attaches itself to IE, thus gaining access to the current page's DOM. The prototype can readily be enabled or disabled using the add-on manager in the IE's *Tools* menu. Note that the integration plug-in does not require any changes to default IE security settings, thus avoiding potential vulnerabilities resulting from lowered browser security settings. Note also that the prototype operate with both the CardSpace and the Higgins[17] identity selectors without any modification.

The prototype has been successfully tested with the 'MyOpenID' OP (https://www.myopenid.com/) and with an experimental implementation of a CardSpace-enabled RP.

[17] `http://wiki.eclipse.org/GTK_Selector_1.1-Win`

5.1 User Registration

Prior to use, the user must have accounts with a CardSpace RP and an OP. The user must also create an IDcard for the relevant OP. This involves invoking the selector and inserting the user's OpenID identifier at the target OP in the *web page* field, the OP URL in the *street* field, and the trigger word *OpenID1.1* or *OpenID2.0* in the *city* field. For ease of identification, the user can give the personal card a meaningful name, e.g. of the target OP site. The user can also upload an image for the card, e.g. containing the logo of the intended OP or simply of OpenID. When a user wishes to use a particular OP, the user simply chooses the corresponding IDcard.

5.2 Prototype Operation

In this section we consider specific operational aspects of the prototype. We refer throughout to the numbered protocol steps given in section 3.2.

In step 3 the plug-in uses the DOM to perform the following processes.

3.1 It scans the web page in the following way[18].

(a) It searches through the HTML elements of the web page to detect whether any HTML forms are present. If so, it searches each form, scanning through each of its child elements for an HTML object tag.

(b) If an object tag is found, it retrieves and examines its type. If it is of type 'application/x-informationCard' (which indicates website support for CardSpace), it continues; otherwise it aborts.

(c) It retrieves and stores in a cookie the name attribute of the CardSpace object tag. This is important since the RP will use this name to retrieve the token from the HTTP POST array.

(d) It searches through the param tags (child elements of the retrieved CardSpace object tag) for the 'issuer' tag and examines its value; if it is '`http://schemas.xmlsoap.org/ws/2005/05/identity/issuer/self`', indicating that the use of personal (self-issued) cards is acceptable, it continues[19]; otherwise it terminates.

[18] The user guide [9] specifies two HTML extension formats that can be used to invoke the selector from a web page, both of which involve placing the CardSpace object tag inside an HTML form. This motivates the choice of the web page search method (see also [2]).

[19] The plug-in also continues if the value of the 'issuer' tag is set to 'any', '*' or if the 'issuer' tag is absent, since the use of personal cards is acceptable in these cases.

(e) It retrieves the 'requiredClaims' and 'OptionalClaims' tags from the param tags, and retrieves and stores the mandatory and optional claim types listed in these tags.

(f) If necessary, and after keeping track of the original policy settings, it modifies the RP policy so that the *city*, *street* and *web page* claim types are specified in the 'requiredClaims' tag.

3.2 It adds a JavaScript function to the head section of the HTML page to intercept the XML-based security token (i.e. the RSTR message).

3.3 It obtains the action attribute of the CardSpace HTML form and stores it in a cookie. This attribute specifies the URL address of a web page at the CardSpace RP to which the security token must be forwarded for processing. If the obtained attribute is not a fully qualified domain name address, the JavaScript inherent properties, i.e. *document.location.protocol* and/or *document.location.host*, are used to help reconstruct the full URL address.

3.4 It changes the current action attribute of the CardSpace HTML form to point to the newly created 'interception' function (see step 3.2 above).

In step 8 the plug-in uses the DOM to perform the following steps.

8.1 It intercepts the RSTR message sent by the selector using the added function.

8.2 It parses and extracts certain RSTR contents. If the *city* field contains the word *OpenID1.1* or *OpenID2.0*, the plug-in proceeds; if not, normal operation of CardSpace continues. It reads the *web page* field to discover the user's OpenID identifier, and obtains the OP URL from the *street* field. In addition, all other fields, notably the SAML assertion ID and the PPID, are parsed and stored in cookies.

8.3 It constructs an OpenID authentication request, compatible with the OpenID version indicated by the trigger word in the *city* field. The plug-in defaults to creating an OpenID 1.1-compatible authentication request if no version is specified. This involves generating a nonce and time-stamp, and also determining the required and optional attributes to be sent to the OP. The plug-in retrieves all the CardSpace-supported claims it stored earlier (see step 3.1 (e) above). It then maps between them and the SREG-supported attributes, using Table 1. The

mapping is done using JavaScript regular expressions, specifically the 'match' method with its global (g) and case-insensitive (i) parameters. The plug-in uses the OpenID 'checkid_setup' mode, and skips the optional initiation phase in which the OP-RP exchange a shared secret. It also sets the return page (to which the OP sends the authentication response) to equal the currently-visited RP page.

8.4 It redirects the user to the OP along with the OpenID authentication request, using the JavaScript inherent property 'window.location'.

In step 11 the plug-in performs the following steps.

11.1 It parses the OP-issued authentication response (embedded in the URL).

11.2 It (transparently) validates the authentication response, including verifying that the return URL (openid.return_to) is the same as the current page, checking the nonce and time-stamp, and validating the OP MAC value on the authentication assertion. The plug-in uses the OpenID 'check_authentication' mode so that the MAC verification is performed by the OP via an TLS/SSL channel; it issues an HTTPS request to the OP with exact copies of all fields from the authentication response (except for the 'openid.mode' field whose value the plug-in changes from 'id_res' to 'check_authentication'). The OP responds with a boolean value of either 'true' or 'false'. If all of the checks succeed, the plug-in continues to the next step; otherwise it terminates, informing the user that the process can no longer continue.

11.3 It constructs a CardSpace-compatible SAML security token, inserting the user attributes received from the OP into the token. It retrieves other token-specific data (including the PPID and the token assertion ID) that was originally contained in the SIIP-issued RSTR from the cookies created earlier (see step 8.2). It also embeds the signed SIIP-issued RSTR into the SAML token.

11.4 It creates and appends an 'invisible' HTML form, with the method attribute set to 'POST', to the current page, thereby delivering the SAML token to the RP.

11.5 It writes the entire SAML security token as a hidden variable into the invisible HTML form, with the name attribute of this variable set to the CardSpace object tag's name (see step 3.1 (c)). Note that the plug-in retrieves this name from the appropriate cookie.

11.6 It writes the end-point URL of the CardSpace-enabled RP into the action attribute of the invisible form. Note that the plug-in retrieves this name from the appropriate cookie (see step 3.3).

11.7 Finally, it auto-submits the HTML form (transparently to the user), using the JavaScript inherent method 'submit'.

5.3 Potential Issues

The integration plug-in must scan every HTML web page to see whether it supports CardSpace, and this may affect system performance. However, informal tests on the prototype suggest that this is not a serious issue. In addition, the plug-in can be configured so that it only operates with certain websites.

If the web browser is compromised, then an adversary could steal the user token (see above), block the user-RP connection, and submit the token, thus impersonating the user. Moreover, if the RP does not use https, then the SIP-issued RSTR will not be encrypted. Assuming that the web browser is not a secure environment, it may be possible for a malicious plug-in or other malware to get access to sensitive information disclosed by the plaintext RSTR and/or the user token. However, the same risks apply when manually entering credentials (e.g. username-password) into the browser [7].

Finally note that some older browsers (or browsers with scripting disabled) may not be able to run the integration plug-in, as it was built using JavaScript. However, most modern browsers support JavaScript (or ECMAscript), and hence building the prototype in JavaScript is not a major usability obstacle.

6 Related Work

A somewhat similar scheme [1] has previously been proposed to support CardSpace-Liberty interoperation. However, unlike the scheme proposed here, the CardSpace-Liberty integration scheme is not transparent to the IdPs, does not support the exchange of identity attributes, and does not support HTTPS-enabled websites.

Kim et al. [5] have proposed an OpenID authentication method using an identity selector. This scheme is designed to reduce phishing and hacking risks, and also simplify user authentication by automatically performing the OpenID-based login process without the need to manually input the OpenID URL. The scheme uses a specially modified identity selector to

enable OpenID authentication, unlike the scheme proposed here which uses an unmodified selector.

Microsoft and OpenID have announced plans[20] to enable a level of interoperation. A stated aim of this effort is to reduce the risk of phishing in OpenID by enabling an OpenID user to employ CardSpace when authenticating to an OP. The scheme proposed here inherently provides a level of protection against phishing since the redirect step to the OP is initiated by the browser extension (see section 4.1), and also supports use of CardSpace to authenticate to OPs.

7 Conclusions and Future Work

We have proposed a means of interoperation between two leading identity management systems, namely CardSpace and OpenID. CardSpace users are able to obtain an assertion token from an OpenID identity provider, the contents of which can be processed by a CardSpace-enabled relying party. The scheme is transparent to OpenID providers and identity selectors, uses a browser extension, and requires only minor changes to a CardSpace-enabled relying party. It uses the CardSpace identity selector interface and CardSpace personal cards to enable interoperation between OpenID providers and CardSpace relying parties.

The integration scheme takes advantage of the similarity between the OpenID and the CardSpace frameworks, and this should help to reduce the effort required for full system integration. Also, implementation of the scheme does not require technical co-operation between Microsoft and the OpenID Foundation.

Planned future work includes investigating the possibility of using the CardSpace identity selector to enable access to identity providers of other identity management systems, such as Shibboleth. We also plan to extend the scheme to support CardSpace-enabled relying parties that employ security token services.

Acknowledgements

The first author is sponsored by the Diwan of Royal Court, Sultanate of Oman.

[20] `http://www.guardian.co.uk/technology/blog/2007/feb/07/openidgetsab`

References

[1] Haitham S. Al-Sinani, Waleed A. Alrodhan, and Chris J. Mitchell. CardSpace-Liberty integration for CardSpace users. In Ken Klingenstein and Carl M. Ellison, editors, *Proceedings of the 9th Symposium on Identity and Trust on the Internet, (IDtrust'10), Gaithersburg, Maryland, USA, April 13–15, 2010*, pages 12–25. ACM, New York, NY, 2010.

[2] Haitham S. Al-Sinani and Chris J. Mitchell. Using CardSpace as a password manager. In Elisabeth de Leeuw, Simone Fischer-Hübner, and Lothar Fritsch, editors, *Proceedings of IFIP IDMAN 2010 — the second IFIP Conference on Policies and Research in Identity Management, November 18–19, 2010, Oslo, Norway. Volume 343 of IFIP Advances in Information and Communication Technology*. Springer, Boston, 18–30, 2010.

[3] Andreas Berger. *Identity Management Systems — Introducing Yourself to the Internet*. VDM Verlag, Saarbrücken, 2008.

[4] Vittorio Bertocci, Garrett Serack, and Caleb Baker. *Understanding Windows CardSpace: An Introduction to the Concepts and Challenges of Digital Identities*. Addison-Wesley, Reading, Massachusetts, 2008.

[5] Seung Hyun Kim et al. *OpenID Authentication Method Using Identity Selector*. United States, Patent Application Publication, Pub. No. US 2009/0249078 A1, 2009.

[6] Dick Hardt, Johnny Bufu, and Josh Hoyt. *OpenID Attribute Exchange 1.0 — Final*. Sxip Identity and JanRain, 2007. `http://openid.net/specs/openid-attribute-exchange-1_0.html`.

[7] Jonathan Hart, Konstantinos Markantonakis, and Keith Mayes. Website credential storage and two-factor web authentication with a Java SIM. In Pierangela Samarati, Michael Tunstall, Joachim Posegga, Konstantinos Markantonakis, and Damien Sauveron, editors, *Proceedings, Information Security Theory and Practices. Security and Privacy of Pervasive Systems and Smart Devices, 4th IFIP WG 11.2 International Workshop, WISTP 2010, Passau, Germany, April 12–14, 2010*, volume 6033 of *Lecture Notes in Computer Science*. Springer, Berlin, Heidelberg, 229–236, 2010.

[8] Josh Hoyt, Jonathan Daugherty, and David Recordon. *OpenID Simple Registration Extension 1.0*. JanRain and VeriSign, 2006. `http://openid.net/specs/openid-simple-registration-extension-1_0.html`.

[9] Michael B. Jones. *A Guide to Using the Identity Selector Interoperability Profile V1.5 within Web Applications and Browsers*. Microsoft Corporation, 2008.

[10] Michael B. Jones, Michael McIntosh, and (editors). *Identity Metasystem Interoperability Version 1.0 (IMI 1.0)*. OASIS Standard, 2009. `http://docs.oasis-open.org/imi/identity/v1.0/identity.html`.

[11] Marc Mercuri. *Beginning Information Cards and CardSpace: From Novice to Professional*. Apress, New York, 2007.

[12] Joaquin Miller and (editor). *Yadis Specification — Yadis 1.0 (HTML)*, 2006. `http://yadis.org/wiki/Yadis_1.0_%28HTML%29`.

[13] OpenID Community. *OpenID Authentication 2.0 — Final*, 2007. `http://openid.net/specs/openid-authentication-2_0.html`.

[14] David Recordon and Brad Fitzpatrick. *OpenID Authentication 1.1*, 2006. `http://openid.net/specs/openid-authentication-1_1.html`.

[15] Michal Zalewski. *Browser Security Handbook*. Google, 2008. `http://code.google.com/p/browsersec/wiki/Main`.